Attorney Docket No.: 10559/535001 / P12545

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: CONTEXT EXECUTION IN A PIPELINED COMPUTER PROCESSOR

APPLICANT: JOHN A. WISHNEUSKY

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL870691083US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, 2900 Crystal Drive, Arlington, VA 22202-3513.

November 19, 2001
Date of Deposit

Signature

Gabe Lewis
Typed or Printed Name of Person Signing Certificate

CONTEXT EXECUTION IN A PIPELINED COMPUTER PROCESSOR

BACKGROUND

This invention relates to scheduling contexts in a pipelined computer processor.

Instruction execution in a computer processor may be accomplished in a variety of ways. For example, a software program that includes multiple instructions may be partitioned into multiple sets of instructions ("contexts") where each context may be "swapped" in or out of execution according to a scheduling system associated with the processor.

DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a block diagram of a pipelined processor system that executes multiple contexts; and

FIG. 2 is a flowchart that depicts a process for executing multiple contexts.

DESCRIPTION

A computer processor may execute individual instructions within a context in a pipelined fashion ("pipelining"). That is, individual instructions are executed in multiple pipeline stages that relate to different logic blocks within the

processor. Pipelining often causes data dependencies, where a first instruction is modifying data that is required by a subsequent instruction, and which may cause the computer processor to stall waiting for the data to become available. Furthermore, pipelined execution of conditional instructions, i.e., instructions that can alter the sequential execution of instructions, (e.g., branches and subroutines) may cause the wrong instructions to be fetched. Scheduling of context execution in a pipelined processor affects the availability of computer processor resources and overall processor performance.

Referring to FIG. 1, a computer processing system 10 that executes multiple contexts includes a computer processing unit (CPU) 20 that executes instructions in a pipelined fashion, an instruction memory 30 for holding instructions to be executed by CPU 20 and a set of executing context registers (ECR) 40a-40d for holding context information for contexts that have been scheduled for execution. ECR 40a-40d are storage "registers", each storing an instruction address for the next instruction to be executed by each context, respectively. System 10 also includes an instruction fetch/execution control block 50 ("control block 50") for selecting instruction addresses from ECR 40a-40d that are input to instruction

memory 30 and for selecting instructions from ECR 40a-40d that are input to CPU 20. System 10 also includes a context scheduler 60 that is connected to ECR 40a-40d by new context bus 65. Context scheduler 60 stores context information in ECR 40a-40d for contexts determined ready for execution by scheduler 60, as will be explained.

CPU 20 is connected to send control information 25 related to the execution of previous instructions to each of ECR 40a-40d. Control information 25 may include an indication that a context "exit" instruction was executed, or an indication of the execution of a conditional instruction, or the computed instruction address from a previously executed conditional instruction. Control information bus 25 is also connected to scheduler 60 to indicate the execution of a context exit instruction.

System 10 executes up to four (4) contexts concurrently, executing a single instruction in sequence from each context in ECR 40a-40d. Control signals 55 from control block 50 to ECR 40a-40d selects one instruction from each context every fourth execution cycle of CPU 20. By executing only a single instruction from each context 40a-40d, respectively, all of the pipeline stages of each instruction are able to complete

before the execution of any subsequent instructions in the same context 40a-40d, respectively.

Therefore, "results" from the execution of a first instruction in a context will be available before a second instruction in the same context might reference those results. The results may include, for example, an ALU computation, a completion of a data read and a completion of a data write. Also, the results may include instruction-fetching decisions, for example, a computed address for a branch or subroutine instruction. The computed address is used to fetch the "correct" instructions, that is, avoiding the sequential fetching of instructions that may not need to be executed.

ECR 40a-40d are connected to receive a computed address from CPU 20 on control information bus 25 and also connected to receive an initial instruction address for new contexts to be executed on bus 65 from scheduler 60. ECR 40a-40d also store the actual instruction for execution, after the instruction is loaded from instruction memory 30 on instruction bus 31. ECR 40a-40d may also store additional information relating to the execution state required to resume execution of the stored instruction by CPU 20, e.g., CPU state and register values required to properly resume execution of a context.

In operation, at system start-up, context scheduler 60 loads an instruction address into each ECR 40a-40d on bus 65. Thereafter, during operation, whenever an ECR 40a-40d becomes available, context scheduler 60 loads another instruction address for another context determined ready for execution. In a multiple context execution system, such as system 10, contexts eventually complete and must be 'de-scheduled'. In system 10 de-scheduling occurs upon execution of a 'context exit' instruction contained within a context.

However, in order to avoid the fetching of instructions for the context containing the context exit instruction, the context exit instruction must be recognized by CPU 20 and indicated to scheduler 60. More specifically, when a context exit instruction is executed by CPU 20 control information is sent on bus 25 to scheduler 60 to indicate the completion of a context and the availability of an ECR 40a-40d. In response, scheduler 60 will cause a new instruction address for a new context to be loaded into the available ECR 40a-40d. Many scheduling schemes can be used to determine when a new context is ready for execution.

Control logic 50 operates as a "distributor" of selection counts and timing pulses. That is, control logic 50 distributes values from a repeating selection counter and a

timing pulse, on output lines 55, 53 and 51, to each of ECR 40a-40d, address mux 32 and instruction mux 22, respectively. More specifically, control logic 50 outputs a selection count to each of address mux 32, instruction mux 22 and each of ECR 40a-40d in a repetitive "control cycle".

For example, during a first "control cycle", control logic 50 outputs a selection count 53 that selects the instruction address from ECR 40a being input to address mux 32 causing an output of an instruction from instruction memory 30 that is stored in ECR 40a. During the same control cycle, control logic 50 also outputs a selection count 51 that selects an instruction from instruction mux 22 from ECR 40b that causes the input of the selected instruction to CPU 20. Also, during the same control cycle, control logic 50 outputs a selection count to ECR 40d to load control information 25 caused by a previously executed instruction from the context of ECR 40d. During the next control cycle, control logic 50 will increment the selection counts on lines 55, 53 and 51 to repeat the sequence of loading instructions, addresses and control information to and from ECR 40a-40d.

The use of four (4) ECR 40a-40d in system 10, and the sequential selection of each ECR 40a-40d in a round-robin fashion by control block 50 allows enough time between

instruction execution of each context for control information to be available from a previous instruction executed by CPU 20. However, other numbers of ECRs could be used to provide more or less time between instruction execution and to accommodate more or less pipeline execution over-lap in CPU 20.

System 10 operates most efficiently with software applications that are readily partitioned into multiple contexts, for example, network processing applications each of which interact with different hardware blocks or interfaces connected to system 10, or the like. Therefore, applications that allow the scheduling of four (4) contexts will maximize the use of available processing cycles of CPU 20. However, even applications that allow less than four (4) contexts to be concurrently scheduled will still be able to gain some efficiency in a pipelined CPU 20, that is, since each instruction executed in system 10 in a round-robin fashion may still be provided with additional time for the control information 25 to become available from a first instruction before execution of a subsequent instruction.

A known way of handling data dependencies issues in a pipelined processor is to provide a "bypass bus" and associated control logic. The bypass bus is a data bypass

line from the output to the input of an arithmetic logic unit that is used to provide data modified by a first instruction directly to the input of the ALU for used by a subsequent instruction requiring the data. In an integrated processor, including a bypass bus and associated control logic adds complexity to the design and occupies logic area that could be used for other purposes. By contrast, in system 10, the context scheduling and execution control allows for a far simpler CPU design, i.e., a CPU that does not require an ALU bypass bus and associated logic. More specifically, in system 10, ALU and register operations for a first instruction executed in CPU 20 will complete and data will be available before a second instruction in the same context requires the data.

As described previously, CPU 20 does not require an ALU bypass bus and associated bypassing logic to deal with pipelining issues relating to data dependencies or conditional instruction execution. However, CPU 20 could include these features.

Referring to FIG. 2, a process 70 of executing multiple contexts in a computer processor includes storing 72 instruction addresses for contexts ready for execution in execution context registers (ECRs). The process 70 selects 74

a first instruction address from a first ECR and selects 74 an instruction for execution from a second ECR in the same execution cycle. The process 70 determines 76 control information and/or a computed address from a previous instruction execution, and determines 78 whether the previous instruction was a 'context exit' instruction. If the previous instruction was a context exit, process 70 includes storing 80 a new context for execution in an available ECR and repeating the sequence of actions 74-78 for other contexts stored in the ECRs. If the previous instruction was not a context exit, process 70 includes selecting and loading 82 the control information and/or the computed address into a third ECR, and repeating the sequence of actions 74-78 for other contexts stored in the ECRs.

In an embodiment, process 70 is implemented in a computer program that includes instructions that are executed by a computer processor.

A number of embodiments of the invention have been described. Other embodiments are within the scope of the following claims.